2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 24–28, 2017, Vancouver, BC, Canada

Behaviour-Data Relations Modelling Language For Multi-Robot Control Algorithms*

Lenka Pitonakova[1,3], Richard Crowder[1] and Seth Bullock[2]

Abstract— **Designing and representing control algorithms is challenging in swarm robotics, where the collective swarm performance depends on interactions between robots and with their environment. The currently available modeling languages, such as UML, cannot fully express these interactions. We therefore propose a new, Behaviour-Data Relations Modeling Language (BDRML), where robot behaviours and data that robots utilise, as well as relationships between them, are explicitly represented. This allows BDRML to express control algorithms where robots cooperate and share information with each other while interacting with the environment.**

I. INTRODUCTION

Having a suitable modeling language is essential for designing, representing and reproducing software, including that which is written for robots. Various methods are currently used to describe behaviour in collective robotics. While some authors prefer to use text alone [1], [2], [3], [4], others also rely on visual representations, such as statecharts [5], [6], [7], class diagrams [8], [9], or sequence charts [9], [10], that follow grammar of formalised modeling languages.

As we will demonstrate below, a major drawback of these visualisation methods is that they do not represent information explicitly. Particularly in collective robotics, a "bottom-up" approach to behaviour design is required [11], meaning that control algorithms of individual robots need to be programmed, but that the collective performance emerges as a result of complex robot-robot and robot-environment interactions, during which robots acquire, share and process information [10], [12], [13]. Consequently, algorithm representations that do not fully take data representations into account are often ambiguous and non-comprehensive.

In this paper, the *Behaviour-Data Relations Modeling Language* (BDRML) is defined. The language facilitates unambiguous visual and textual representation of robot behaviours, as well as of their interactions with various types of data structures. Unlike in other modeling languages, conditional relationships between robot behaviour and data structures are expressed explicitly in BDRML, allowing the language to represent control algorithms where robots cooperate and share information with each other and where they acquire and store information in their environment.

*This work was supported by an EPSRC Doctoral Training Centre grant (EP/G03690X/1) and by Thales UK

[1]Dep. of Electronics and Computer Science, University of Southampton, United Kingdom
[2]Dep. of Computer Science, University of Bristol, United Kingdom
[3]contact@lenkaspace.net

II. BACKGROUND

The use of natural language [1], [2] or pseudocode [3], [4] to describe robot behaviour, as well as the way in which robots exchange information, is popular. However, while textual descriptions are often comprehensive, the reader is required to process a lot of text in order to understand the algorithm described. Furthermore, important parts of the robot program and key data structures are difficult to identify.

On the other hand, visual representations can provide a quick overview that is often more accessible than text. A popular visualisation method is a statechart, where robots are represented as finite-state machines with "states" indicated in boxes and "state transitions" shown as arrows between the boxes [5], [6], [7] (Figure 1a). State transitions can either be based on boolean conditions or probabilities, that are usually defined in equations outside of the diagram.

A visual representation of behaviour is useful not only for an algorithm description, but also during its creation. Therefore, design patterns, that provide "templates" for algorithm design [14], are often accompanied by diagrams. The most popular visualisation method for design patterns, both in object-oriented software and in multi-agent engineering, is a class diagram [8], [9] (Figure 1b). Both statecharts and class diagrams are part of the Unified Modeling Language (UML) [15].

The utility of statecharts and class diagrams for modelling multi-agent systems, such as robot swarms, is limited for two main reasons. Firstly, data in these diagrams is not represented explicitly, making it difficult to express where information is stored or how it is operated on. It has already been demonstrated that information processing is an important part of swarm behaviour [10], [12], [13] and an adequate representation of data is thus essential when describing swarm algorithms. Secondly, swarm control algorithms often rely on cooperation or communication between robots. Therefore, a way of representing relationships between behaviours and data of two different robots is needed. Because of these drawbacks, statecharts and class diagrams often have to be complemented either by textual description of how robot state transitions depend on data acquired from other robots [6], [7], [16] or by a different type of a diagram that specifically represents communication, such as a sequence chart [9], [10] (Figure 1c).

Some authors address this problem by extending UML or by creating custom diagrams that do not belong to a specific modeling language. Such representations are often intuitive enough to understand, but since their rules are

978-1-5386-2681-8/17/$31.00 ©2017 IEEE

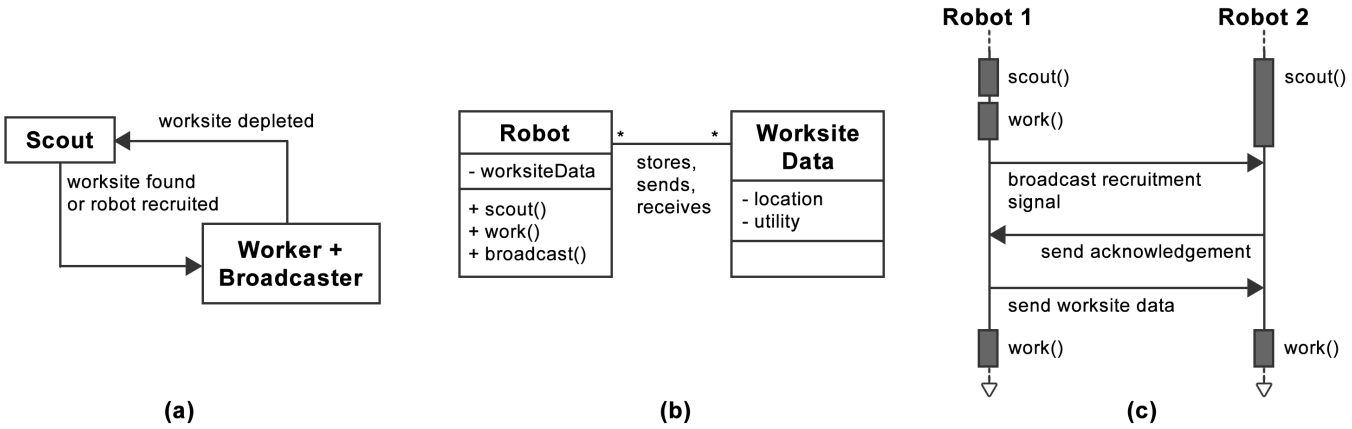


Fig. 1. Visual representation of a collective task allocation robot control algorithm in the form of (a) statechart, (b) class diagram, (c) sequence chart. The robot searches the environment as a "Scout" and becomes a "Worker" upon finding a worksite. A Worker extracts resources from the worksite, while broadcasting recruitment signals to Scouts that are nearby. Any Scout that is recruited becomes a Worker as well. A Worker resumes scouting when its worksite is depleted.

not clearly defined, they can be ambiguous and difficult to apply to different control algorithms [17]. For example, in statecharts for ant-inspired swarms, textual descriptions inside of state boxes and above state transition arrows have been used to indicate when pheromone is used by robots [4], [6]. Class diagrams have been extended in a similar fashion in order to show entities such as "agent" and "gradient field", with arrows between them representing data transfer [8]. Similarly, in other customised diagrams, different robot types and data storage devices have been visualised as boxes, with arrows indicating data exchange between them [18], [19]. It is important to point out that all these custom diagrams, with the exception of those in [4], represent *communication behaviour*, such as "notify" or "read", but do not explicitly visualise relationships between specific data structures and robot behaviours. Therefore, they still need to be accompanied by a considerable amount of text that clarifies what data is communicated and how it affects robot behaviour.

III. BDRML PRIMITIVES

The *Behaviour-Data Relations Modelling Language* (BDRML) proposed here addresses the shortcomings of the existing modelling languages in the following way. It defines a set of *primitives*, that represent robot behaviours and data, and a set of conditional *relations* between these primitives. All these elements have their visual as well as textual representations, which can be used together or separately, and can fully describe most collective algorithms with minimal need for an additional explanation in natural language.

There are three types of primitives in BDRML (Figure 2):

- ***Behaviour***, i.e., a set of processes that deal with a particular situation that a robot finds itself in, for example "Scout" or "Rest"
- ***Internal data structure***, i.e., information that is stored in a robot's memory
- ***External data structure***, i.e., information that is stored in a non-robot entity, for example, in an RFID tag or in a gradient field in the environment

Data structure type always follows the structure's name and a colon in a textual description. Data type names appropriate for the context in which the BDRML diagram is used, e.g., boolean, int, float, object, etc. may be used.

Note that "behaviours" in BDRML, such as "Work" or "Scout", can refer to "states" or sets of "states" in finite-state machines. In neural network controllers, "behaviours" need not be programmed explicitly, but would manifest through the network dynamics.

Also note a crucial difference between internal and external data. Internal data is readily available to a robot at any given point in time, while external data has to be obtained from the environment. Moreover, when information needs to be exchanged between robots, data stored internally can only be passed from one robot to another when they are within each other's communication range. On the other hand, external data can be deposited by one robot into the environment and read by another robot later.

Since both behaviours and data are primitives, BDRML al-

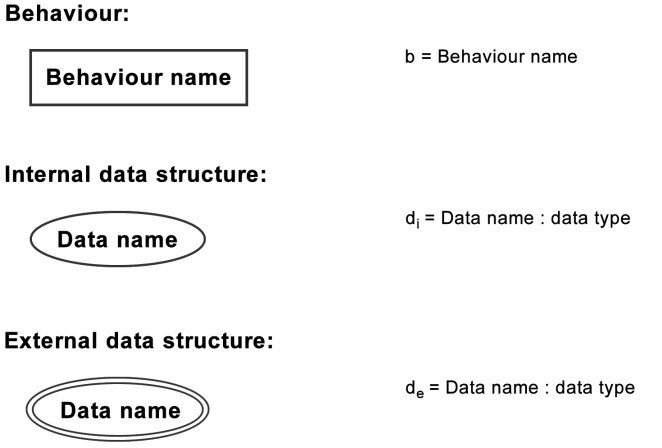


Fig. 2. Visual (left) and textual (right) representation of the BDRML primitives.

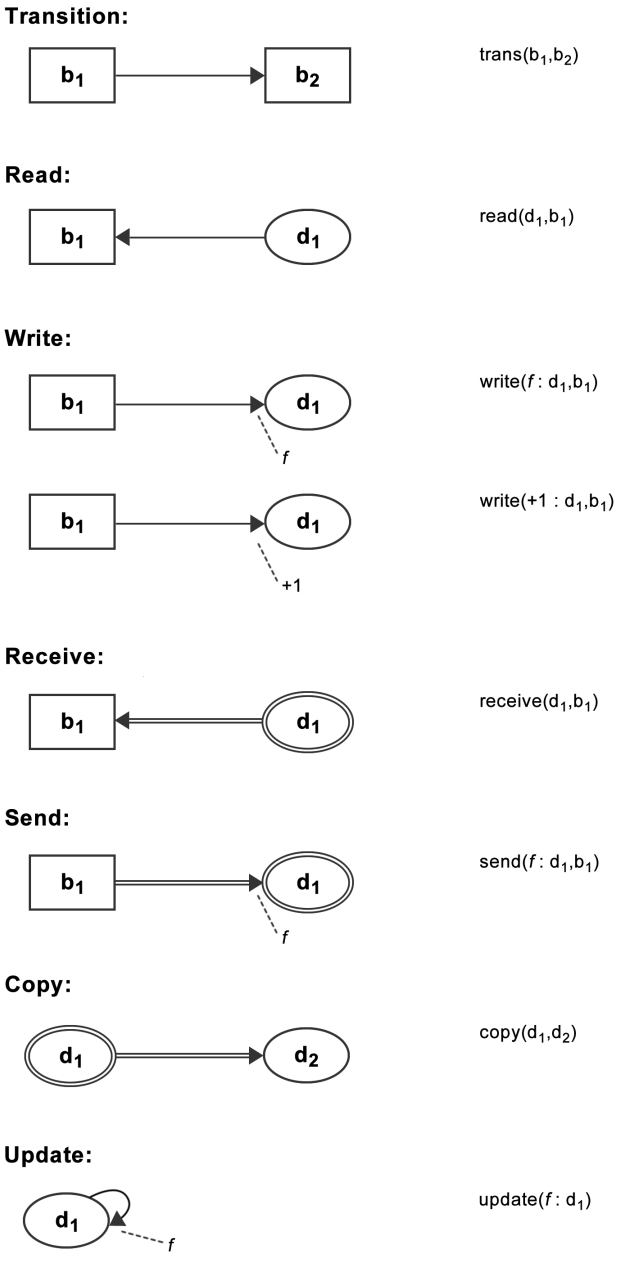


Fig. 3. Visual (left) and textual (right) representation of the BDRML relations.

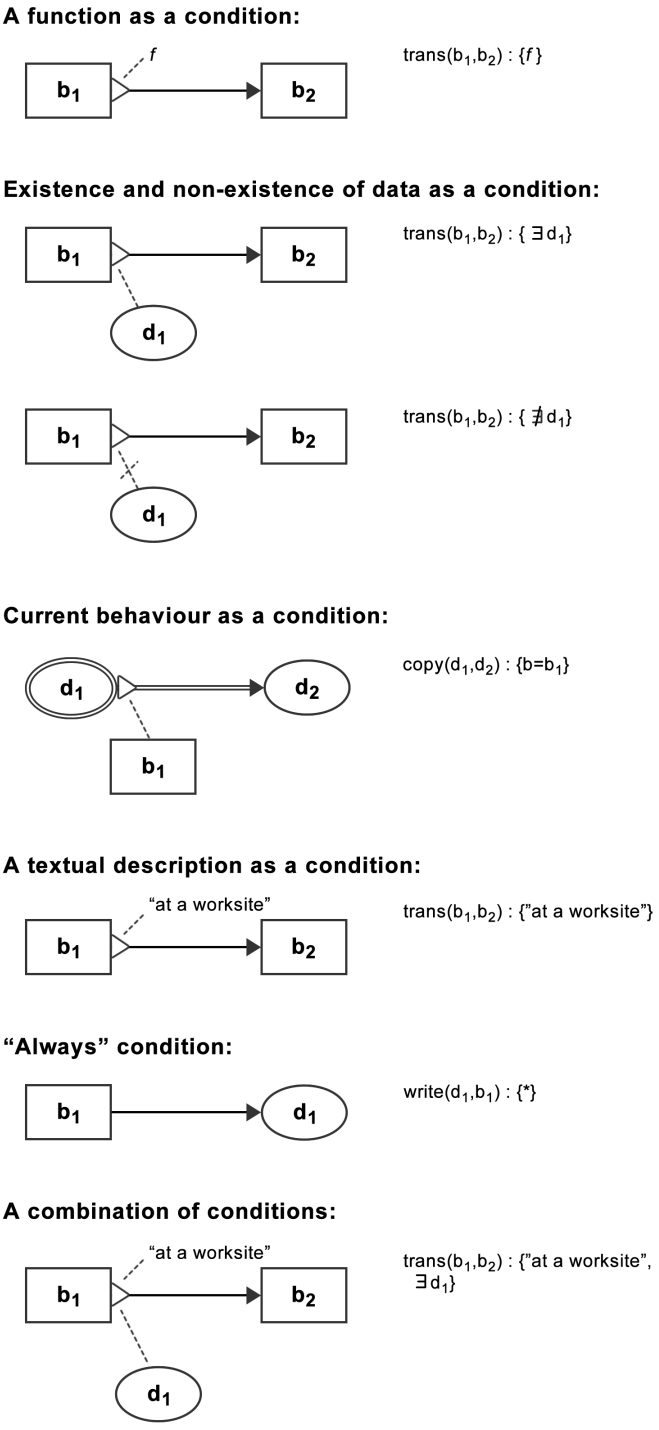


Fig. 4. Visual (left) and textual (right) representation of the BDRML conditions.

lows precise specification of relations between robot actions and information. There are seven types of relations possible (Figure 3):

- ***Transition:*** a behaviour-behaviour relation, where the robot transitions from one behavioural mode to another
- ***Read:*** a behaviour-data relation, where *internal* data, stored in the robot's memory, is used by the robot when it is engaged in a particular behaviour
- ***Write:*** a behaviour-data relation, where an *internal* data structure is written into when a robot is engaged in a particular behaviour. Optionally, the new value or a function that defines it can be indicated next to a dashed line extending from the end of the relation arrow in the visual description, and written before a colon proceeding the data structure name in a textual description.
- ***Receive:*** a behaviour-data relation, where *external* data, stored in the environment, is used by the robot when it is engaged in a particular behaviour
- ***Send:*** a behaviour-data relation, where *external* data

is stored in the environment by the robot when it is engaged in a particular behaviour. Alternatively, the robot sends data to another robot that stores the data as *internal*. As is the case with the *write* relation, the new value or a function that defines it may optionally be specified.

- ***Copy:*** a data-data relation, where information is copied from one data primitive to another, for example, from an external to an internal data structure that represents the same information
- ***Update:*** a relation of a data structure with itself, where its value is updated from that in the previous time step by a subroutine not visualised in the BDRML diagram (for example, a robot's hunger level might "spontaneously" increase by one at every time step). The new value or a function that defines it *must* be specified.

Note that it is assumed that a relation is applied once per time step. For example, a "+1" write relation means that value of a particular variable is increased by one in each time step of the program that a BDRML diagram represents.

It is also necessary to define a set of *conditions* under which a particular relation may occur. A condition is visually represented as an annotated triangle at the beginning of a relation arrow. In a textual representation, a condition set follows a relation signature and is separated from it by a colon (Figure 4). A condition may be annotated as a name of a boolean function or a probability, as an existence or a non-existence of a data structure, as robot being engaged in a certain behaviour (relevant, e.g., in the case of the "copy" relation), or as a simple and unambiguous textual description. A special type of condition is an "always" condition, represented by an asterisk (*). Visually, a relation with an "always" condition may be represented without the condition triangle symbol. Multiple conditions can affect a single relation. Unless otherwise specified, the "or" logical operator is assumed when conditions are combined.

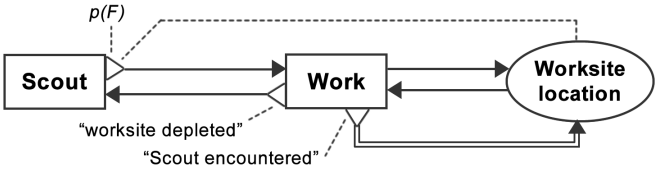
Note that there are three types of lines used in BDRML. Single solid lines represent transitions between behaviours and read/write relations between behaviours and internal data structures. Double solid lines represent some form of communication and link external data structures with behaviours (e.g., in the case of the "receive" relation) and with internal data structures (e.g., in the case of the "copy" relation). Double solid lines can also link a behaviour with an internal data structure during the "send" relation, signifying that a robot engaged in a particular behaviour sends information to another robot, that stores it in its own memory. Finally, dashed lines are used for annotating relation details and conditions.

IV. EXAMPLES

A full BDRML representation consists of both visual and textual specification. A set of behaviours, B, internal data structures, D_i and external data structures, D_e, are first defined, followed by a list of relations between them. Each box, circle and arrow in the visual representation must have a corresponding element or line in the textual representation and vice versa. An example is shown in Figure 5. The described algorithm allows robots to search for worksites and recruit each other to perform work and it can be applied for decentralised task allocation [1]. UML and sequence chart representations were shown in Figure 1. A robot performs the "Scout" behaviour by searching the environment for worksites that can be found with a probability $p(F)$. A successful Scout, that finds a worksite, performs the "Work" behaviour, during which it reads from and writes into its internal data structure, "Worksite location", to keep track of where the worksite is located. Additionally, a working robot sends Worksite location to any Scout that it encounters in order to recruit it. Note how the condition that allows a robot to transition from the Scout to the Work behaviour can be triggered by both $p(F)$ or by recruitment, i.e., by existence of the internal data structure in the Scout's memory. Also note that the condition of recruitment, "scout encountered" signifies that the two robots have to be at a similar place at a similar time for recruitment to occur. The BDRML diagram fully and unambiguously describes when recruitment is performed, what information is exchanged between robots and how it affects robot behaviour.

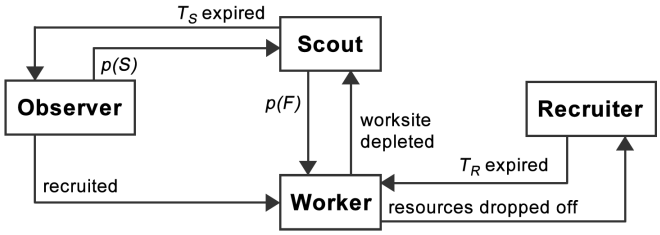
A more complex example of a bee-inspired robot control algorithm for collective foraging, where robots drop off resource and recruit each other in the base [13], is shown as a statechart in Figure 6a and as a BDRML diagram in Figure 6b. As in the previous example, a "Scout" can find worksites in the environment with a probability $p(F)$, after which it starts working by loading resources from a worksite, dropping them in the base and returning to the worksite until it is depleted. After resource has been dropped off in the base, the robot recruits Observers that are present in the base for a certain amount of recruitment time, T_R. Any Scout that cannot find worksites within a certain scouting time, T_S, returns to the base and becomes an Observer. When an Observer is recruited, it becomes a Worker. Alternatively, an Observer transitions back to being a Scout with a scouting probability, $p(S)$.

Note again how all relevant data structures are represented

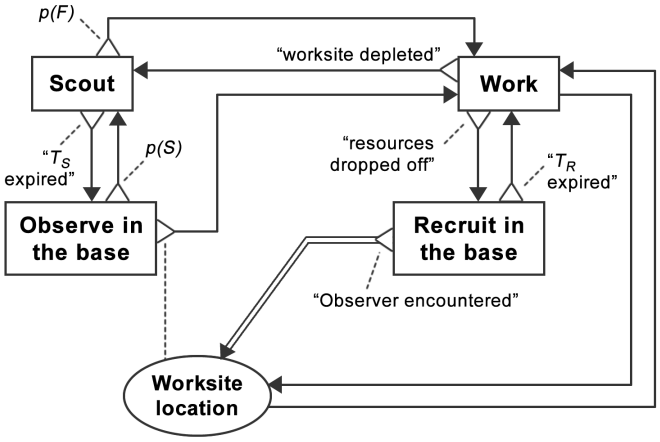


B = {Scout, Work}
D_i = {Worksite location : object}
trans(Scout, Work) : {*p(F)*, ∃ Worksite location}
trans(Work, Scout) : {"worksite depleted"}
write(Worksite location, Work) : {*}
read(Worksite location, Work) : {*}
send(Worksite location, Work) : {"Scout encountered"}

Fig. 5. BDRML representation of the robot control algorithm depicted in Figure 1



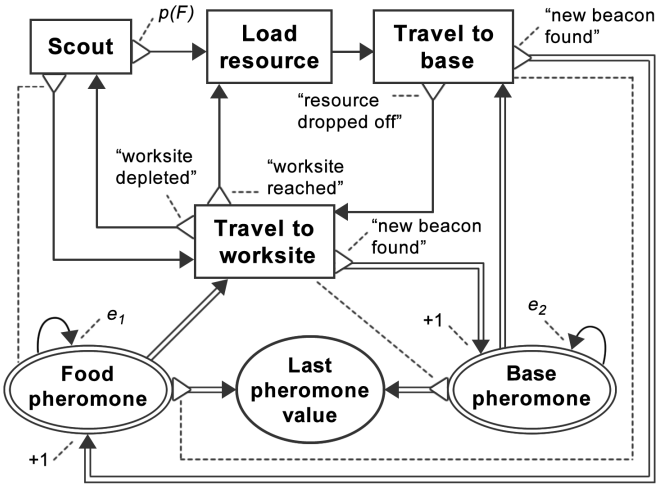
(a)



B = {Scout, Observe in the base, Work, Recruit in the base}
D_i = {Worksite location: object}
trans(Scout, Work) : {*p(F)*}
trans(Work, Scout) : {"worksite depleted"}
write(Worksite location, Work) : {*}
read(Worksite location, Work) : {*}
trans(Scout, Observe in the base) : {"T_S expired"}
trans(Observe in the base, Scout) : {*p(S)*}
trans(Observe in the base, Work) : { ∃ Worksite location}
trans(Work, Recruit in the base) : {"resources dropped off"}
trans(Recruit in the base, Work) : {"T_R expired"}
send(Worksite location, Recruit in the base) : {"Observer encountered"}

(b)

Fig. 6. Representation of a bee-inspired foraging robot control algorithm as a (a) statechart, (b) BDRML diagram



B = {Scout, Load resource, Travel to base, Travel to worksite}
D_i = {Last pheromone value : int}
D_e = {Food pheromone : int, Base pheromone : int}
trans(Scout, Load resource) : {*p(F)*}
trans(Scout, Travel to worksite) : { ∃ Food pheromone}
trans(Load resource, Travel to base) : {*}
trans(Travel to base, Travel to worksite) : {"resource dropped off"}
trans(Travel to worksite, Load resource) : {"worksite reached"}
trans(Travel to worksite, Scout) : {"worksite depleted"}
send(+1 : Food pheromone, Travel to base) : {"new beacon found"}
receive(Base pheromone, Travel to base) : {*}
copy(Food pheromone, Last pheromone value) : {b=Travel to base}
send(+1 : Base pheromone, Travel to worksite) : {"new beacon found"}
receive(Food pheromone, Travel to worksite) : {*}
copy(Base pheromone, Last pheromone value) : {b=Travel to worksite}
update(e_1 : Food pheromone)
update(e_2 : Base pheromone)

Fig. 7. BDRML representation of an ant-inspired foraging robot control algorithm

in BDRML but cannot be directly expressed in a statechart. For example, it is not clear from the statechart that robots in the "Recruiter" state send information to robots in the "Observer" state. This is only shown implicitly, through the "recruited" annotation above the arrow from "Observer" to "Worker" in Figure 6a. In BDRML, an explicit relationship between "Recruit in the base" and "Worksite location" exists, as well as an explicit condition of transitioning from "Observe in the base" to "Work" behaviour, that involves existence of "Worksite location" in the Observer's memory.

A final example, depicted in Figure 7, shows a BDRML representation of an ant-inspired collective foraging algorithm, where robots utilise "beacons" in the environment in order to build a gradient of virtual pheromone from the base to a worksite and back [20], [21]. While performing the "Travel to Worksite" behaviour, a robot updates the "Base pheromone" that is stored in a beacon nearby, so that its value increases by one each time a new beacon is visited. In order to do this, a robot needs to keep track of the "Last pheromone value" in an internal data structure. Because the robot travels towards the Worksite, a gradient of Base pheromone emerges that can be followed on the way back to the base. Gradient of the "Food pheromone" is built in a similar fashion while the robot travels back to the base. Any robot performing the "Scout" behaviour that encounters a beacon can follow the Food pheromone gradient in order to reach a worksite and start working. Values of both pheromones decrease over time in each beacon according to some evaporation function, e_N.

A statechart of the same algorithm would not be able to clearly represent when pheromone values are read and updated, or that values of pheromones, which are stored outside of the robots, decrease over time. Similar problems would be encountered when using a sequence chart. Since arrows in a sequence chart can only represent active communication, i.e., data transfer *from* robot 1 *to* robot 2, it would not be possible to show how robots read data from the environment, i.e., how information is acquired *by* a robot *from* a passive beacon.

V. SUMMARY AND DISCUSSION

BDRML is a new modeling language created specifically for collective robotics, that can be used to design, represent and develop robot programs and relevant data structures. A BDRML description consists of a visual and a textual representation, that can be used together or separately, and includes primitives, i.e., robot behaviours and internal and external data structures, as well as conditional relations between them. A BDRML representation differs from other representations, such as statecharts, class diagrams and sequence charts in three important ways:

- It describes robot *behaviours*, not *states*. For example, while a statechart may represent a "Worker" state, BDRML represents "Work" behaviour (Figure 5). This allows BDRML to model not only finite-state machines, but also, for example, neural network controllers, behaviour-based controllers, etc.
- It explicitly represents *data* (e.g., "Worksite location" in Figure 5), rather than *communication routines* (e.g., "broadcast recruitment signal" in Figure 1c). BDRML thus combines capabilities of statecharts and class diagrams, which describe a robot control algorithm, and sequence charts, which depict communication between robots, by clearly expressing what data is being transferred and when.
- It can include relations between robot states and data external to a robot's memory. This allows BDRML to provide a more complete overview of the robot swarm and its environment and of interactions between the two.

The last two points are especially important for swarm robotics, where interactions and communication between robots and with their environment need to be considered (see Sections I and II).

In this paper, we have applied BDRML to represent robot control algorithms for collective foraging and task allocation with homogeneous robot swarms. In the future, the language will need to be extended so that additional aspects of robots and their environment, relevant in other swarm applications, can be modelled. For example, in order to represent algorithms for collective construction and sorting, entities that can be manipulated by robots, such as "bricks", will need their unique representations. In representations of heterogeneous robot swarms, behaviours and data structures of different robot types, other programmable entities and humans will need to be clearly distinguished.

Another research avenue, that we are currently pursuing, is applying BDRML to express design patterns for robot swarms. Design patterns represent templates for behaviours that can be combined together in order to create a full robot control algorithm. It is therefore crucial to include formal grammar rules in BDRML according to which multiple design patterns, expressed in BDRML, can be unambiguously combined.

REFERENCES

[1] M. O. F. Sarker and T. S. Dahl, "Bio-Inspired communication for self-regulated multi-robot systems," in *Multi-Robot Systems, Trends and Development*, T. Yasuda, Ed. InTech, 2011, pp. 367–392.

[2] F. Ducatelle, G. A. Di Caro, C. Pinciroli, and L. M. Gambardella, "Self-organized cooperation between robotic swarms," *Swarm Intelligence*, vol. 5, no. 2, pp. 73–96, 2011.

[3] N. Lemmens, S. de Jong, K. Tuyls, and A. Nowe, "Bee behaviour in multi-agent systems," in *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*, K. Tuyls, A. Nowe, Z. Guessoum, *et al.*, Eds. Berlin: Springer, 2008, vol. 4865, pp. 145–156.

[4] N. Hoff, R. Wood, and R. Nagpal, "Distributed colony-level algorithm switching for robot swarm foraging," in *Distributed Autonomous Robotic Systems*, A. Martinoli, F. Mondada, N. Correll, *et al.*, Eds. Berlin: Springer, 2013, vol. 83, pp. 417–430.

[5] J. Wawerla and R. T. Vaughan, "A fast and frugal method for team-task allocation in a multi-robot transportation system," in *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA 2010)*. Piscataway, NJ: IEEE Press, 2010, pp. 1432–1437.

[6] R. Fujisawa, S. Dobata, K. Sugawara, and F. Matsuno, "Designing pheromone communication in swarm robotics: Group foraging behavior mediated by chemical substance," *Swarm Intelligence*, vol. 8, no. 3, pp. 227–246, 2014.

[7] E. Castello, T. Yamamoto, F. D. Libera, W. Liu, F. F. T. Winfield, *et al.*, "Adaptive foraging for simulated and real robotic swarms: the dynamical response threshold approach," *Swarm Intelligence*, vol. 10, no. 1, pp. 1–31, 2016.

[8] T. De Wolf and T. Holvoet, "Design patterns for decentralised coordination in self-organising emergent systems," in *Proceedings of the 4th International Workshop on Engineering Self-Organising Systems (ESOA'06)*, S. A. Brueckner, S. Hassas, M. Jelasity, and D. Yamins, Eds. Berlin: Springer, 2007, vol. 4335, pp. 28–49.

[9] T. T. Do, M. Kolp, and A. Pirotte, "Social patterns for designing multi-agent systems," in *Proceedings of the 15th International Conference on Software Engineering & Knowledge Engineering (SEKE 2003)*, G. Webb and H. Dai, Eds. Skokie, Ill.: Knowledge Systems Insttitute, 2003, pp. 103–110.

[10] J. L. Fernandez-Marquez, G. Di Marzo Serugendo, S. Montagna, M. Viroli, and J. L. Arcos, "Description and composition of bio-inspired design patterns: A complete overview," *Natural Computing*, vol. 12, no. 1, pp. 43–67, 2013.

[11] H. V. D. Parunak and S. A. Brueckner, "Software engineering for self-organizing systems," *The Knowledge Engineering Review*, vol. 30, no. 4, pp. 419–434, 2015.

[12] J. M. Miller, X. R. Wang, J. T. Lizier, M. Prokopenko, and L. F. Rossi, "Measuring information dynamics in swarms," in *Guided Self-Organisation: Inception*, M. Prokopenko, Ed. Berlin: Springer, 2014, vol. 9, pp. 343–364.

[13] L. Pitonakova, R. Crowder, and S. Bullock, "Information flow principles for plasticity in foraging robot swarms," *Swarm Intelligence*, vol. 10, no. 1, pp. 33–63, 2016.

[14] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Indianapolis, IN: Pearson Education, 1994.

[15] Object Management Group, Unified Modeling Language specification. http://www.omg.org/spec/UML/ [Accessed 23 Jun 2016], 2015.

[16] W. Liu and A. F. T. Winfield, "Modelling and optimisation of adaptive foraging in swarm robotic systems," *The International Journal of Robotics Research*, vol. 29, no. 14, pp. 1743–1760, 2010.

[17] D. Harel and B. Rumpe, "Meaningful modeling: what is the semantics of semantics?" *Computer*, vol. 37, no. 10, pp. 64–72, 2004.

[18] J. H. Lee, C. W. Ahn, and J. An, "A honey bee swarm-inspired cooperation algorithm for foraging swarm robots: An empirical analysis," in *Proceedings of the 2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2013)*. Piscataway, NJ: IEEE Press, 2013, pp. 489–493.

[19] D. Zhang, G. Xie, J. Yu, and L. Wang, "Adaptive task assignment for multiple mobile robots via swarm intelligence approach," *Robotics and Autonomous Systems*, vol. 55, no. 7, pp. 572–588, 2007.

[20] B. Hrolenok, S. Luke, K. Sullivan, and C. Vo, "Collaborative foraging using beacons," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, W. van der Hoek, G. A. Kaminka, Y. Lesperance, *et al.*, Eds. Richland, SC: IFAAMAS, 2010, pp. 1197–1204.

[21] N. Hoff, A. Sagoff, R. J. Wood, and R. Nagpal, "Two foraging algorithms for robot swarms using only local communication," in *Proceedings of the 2010 IEEE International Conference on Robotics and Biomimetics (ROBIO 2010)*. Piscataway, NJ: IEEE Press, 2010, pp. 123–130.